**AOP - The Business Problem**

**Aspect-Oriented Programming (AOP)**:

In computing, aspect-oriented programming (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. Aspect-oriented programming entails breaking down program logic into distinct parts (so-called concerns, cohesive areas of functionality).

**Concern**:

A Concern is a term that refers to a part of the system divided on the basis of the functionality. Concerns are two types.

* Core concerns
* crosscutting concerns or system-wide concerns.

**Core concern**:

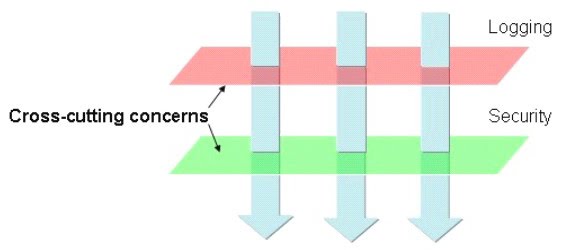
The concerns representing single and specific functionality for primary requirements are known as core concerns or primary functionality of the system is knows as core concerns.

For example: Business logic.

**Crosscutting Concern**:

The concerns representing functionalities for secondary requirements are referred to as crosscutting concerns or system-wide concerns. Or, the crosscutting concern is a concern which is applicable throughout the application and it affects the entire application.

For example: logging, security and data transfer are the concerns which are needed in almost every module of an application, hence they are cross-cutting concerns.



**Aspect**:

An aspect of a program is a feature linked to many other parts of the program, but which is not related to the program's primary function. An aspect crosscuts the program's core concerns, therefore violating its separation of concerns that tries to encapsulate unrelated functions. For example, logging code can crosscut many modules, yet the aspect of logging should be separate from the functional concerns of the module it cross-cuts.

**Keyword**:

AOP -> Aspect Oriented Programming

Aspect -> A Service

Advice -> Service Provider

Point Cut -> A point or condition to execute aspect for business method.

Adviser -> Point Cut with Advice Combination

**New Requirement – Logging, Security**:

* Need to add login and security code to our DAO
* Add some logging statements before the start of the method
* Make sure user is authorized before running DAO method

**Add login and security code**:

public void addAccount(Account theAccount, String userId) {

//add code for loggin

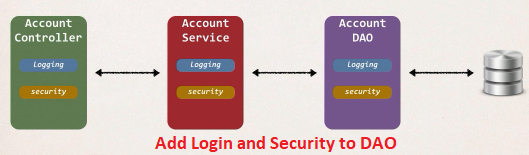
//add code for security check

Session currentSession = sessionFactory.getCurrentSession();

currentSession.save(theAccount);

}

**Let's add to it all to our layers**:



But if we implement this in the above way, we have two main problems.

**Two Main Problems**:

1. **Code Tangling**
   1. For a given method: addAccount
   2. We have logging and security code tangled in
2. **Code Scattering**
   1. If we need to change logging or security code
   2. We have to update ALL classes

**Note**:

Scattering is when code performing a specific function is fragmented and all over the place. Tangling is when a specific piece of code performs multiple functions.

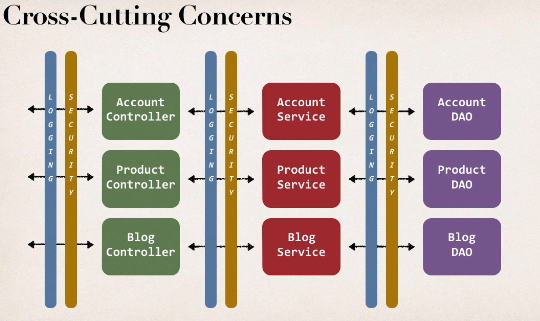
**Other possible solutions**:

1. **Inheritance**
   1. Every class would need to inherit from a base class
   2. Can all classes extend from your base class? Java don't support multiple inheritance
2. **Delegation**
   1. Class would delegate logging, security calls
   2. Still would need to update classes if we want to
      1. Add/Remove logging or security
      2. Add a new feature like auditing, API management, instrumentation

**AOP Aspect-Oriented Programming**:

1. Programming technique based on concept of an Aspect
2. Aspect encapsulates cross-cutting logic
3. “Concern” means logic / functionality

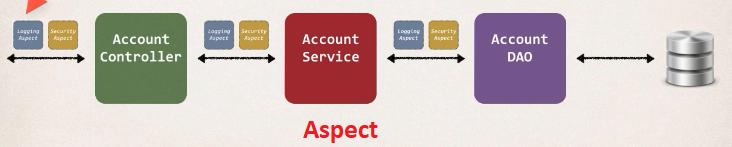
**Cross-Cutting Concern**:



In our project we have to need login and security at all different layers of our application. So, we can basically take our login code, incapsulate it with reusable module or reusable class and we can call this when we need login. Similarly, for security.

**Aspect**:

1. Aspect can be reused at multiple locations
2. Same aspect/class … applied based on configuration



AOP Aspect-Oriented Programming Overview